Group Testing for Image Compression

Edwin S. Hong, Richard E. Ladner

Abstract— **This paper presents Group Testing for Wavelets (GTW), which is a novel embedded wavelet-based image compression algorithm based on the concept of group testing. We explain how group testing is a generalization of the zerotree coding technique for wavelet-transformed images. We also show that Golomb coding is equivalent to Hwang's group testing algorithm (Hwang, 1972). GTW is similar to SPIHT (Said & Pearlman, 1996) but replaces SPIHT's significance pass with a new group testing based method. Although no arithmetic coding is implemented, GTW performs competitively with SPIHT's arithmetic coding variant in terms of rate-distortion performance.**

I. Introduction

Many recent image coding techniques for generating an embedded bit stream rely on coding wavelet coefficients of an image bit-plane by bit-plane, with the most significant bit-plane first. Embedded image coders such as EZW [1], SPIHT [2], and ECECOW [3], differ chiefly in the method of encoding bit-planes. In this paper, we address the problem of how to efficiently code the bit-planes of a wavelet-transformed image.

Shapiro's use of zerotrees in EZW [1] showed that clever coding of wavelet coefficients can lead to image compression algorithms that are fast and effective in rate-distortion performance. SPIHT [2] improved upon EZW with a better method of managing how the trees are subdivided. This paper introduces a new image coder, called Group Testing for Wavelets (GTW). Our algorithm seeks to improve upon these two previous works by placing coefficients into groups which are then coded. Since the groups do not have to be zerotrees, our algorithm can be thought of as a generalization of SPIHT.

Group testing is a technique for identifying a few significant items out of a large pool of items. First studied by Dorfman [4] in 1943, group testing has been applied to many diverse fields. Our observation that zerotree coding is a special case of group testing lead us to investigate group testing as a basis for wavelet coding. In this paper, we describe several group testing techniques and how they apply to data compression. We show that Hwang's group testing algorithm [5] generates a code that is equivalent to an elementary Golomb code [6].

Much of this paper explains our new image compression algorithm (GTW) and its development. Our basic method is divide the coefficients in a bit-plane into different classes, and code each class with a different group tester. We can think of each class of coefficients as a different context, and each group tester as a general entropy coder. Our particular method of defining classes was derived from examining the characteristics of wavelet coefficients of images. We hope the ideas behind GTW's development will lead to a better understanding of how one should code image wavelet coefficients.

We compare the rate-distortion performance of GTW with that of SPIHT (without arithmetic coding) and SPIHT-AC (with arithmetic coding). GTW is significantly better than SPIHT and close to SPIHT-AC over a large range of bit-rates. This is a significant result because GTW does not employ arithmetic coding, but relies only on group testing. It shows that the simple group testing procedure can supplant the need for arithmetic coding in a practical image coder.

This paper is organized as follows: Section II explains some of the basic wavelet image coding techniques. Section III explains the basics of group testing, and its application to data compression. Section IV introduces GTW, the new image compression algorithm. Section V evaluates our algorithm's rate-distortion performance. Section VI discusses the performance of some variations of our algorithm. Finally, we conclude with section VII by comparing GTW with other related work.

II. Wavelet Image Coding Background

A. Basic Techniques of Wavelet Image Coding

Most of the embedded image coders published recently apply a pyramidal wavelet decomposition to an image, and encode the resulting wavelet coefficients in a bit-plane by bit-plane fashion. If we normalize the wavelet coefficients so that their magnitude is less than 1, then bit-plane i would consist of the ith most significant bit of the magnitude of all coefficients written in binary. Thus, the bit-plane ordering represents successive refinement of a simple scalar quantization of the coefficients. We say a coefficient is *significant* when enough bit-planes have been coded so that its value is known to be nonzero. A coefficient is *insignificant* or a *zero coefficient* when it is still indistinguishable from zero.

Wavelet image coding methods typically code each bit-plane with a *refinement pass* and a *significance pass*. The refinement pass for bit-plane i codes those coefficients that were significant in bit-plane $i-1$. The significance pass for bit-plane i codes those coefficients that were insignificant in bit-plane $i-1$; this corresponds to identifying those coefficients that become significant in bit-plane i. This pass also codes the sign of the newly significant coefficients.

A short preliminary version of this paper appeared in Data Compression Conference, DCC 2000, pp. 3-12

University of Washington, Department of Computer Science and Engineering, Box 352350, Seattle, WA 98195-2350. Phone: (206)543-1695. Fax: (206)543-2969. Email: {edhong,ladner}@cs.washington.edu. EDICS: 1-STIL

Research supported by U.S. Army Research Office Grant DAAH004-96-1-0255, NSF grant CCR-9732828, by AT&T, and by Microsoft.

Shapiro, with EZW, pioneered the use of *zerotrees* in its significance pass to predict the insignificance of coefficients. A zerotree is a set of zero coefficients from many different subbands of similar orientation that represent the same spatial location in the image. This set is organized in a tree, where each coefficient not in the lowest frequency subband has four children coefficients in the next higher frequency subband of similar orientation (assuming it exists). These children always represent the same spatial location, as illustrated in figure 1. Coefficients in the lowest frequency subband have children in the subbands of different orientations at the same frequency level. Although Shapiro's EZW and Said and Pearlman's SPIHT both used these trees, the exact assignment of children to the lowest frequency subband was slightly different in the two algorithms.

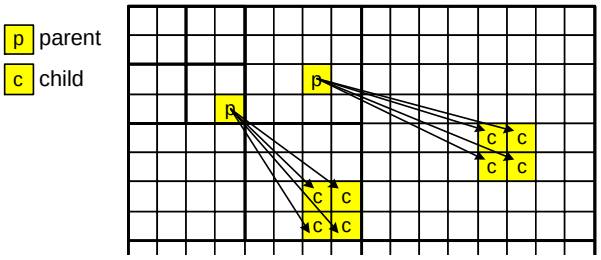

Fig. 1. Illustration of two parents connected to their children in a 3-level wavelet transform

The zerotree method places all the coefficients into large tree structures, and tests whether the trees are *significant*. A tree is significant if any coefficient in that tree is significant; a tree is insignificant if all its coefficients are insignificant (i. e., if it is a zerotree). Each tree is coded with a single symbol. Only the significant trees must be subdivided into smaller subtrees which will then need to be tested for significance and coded; the values of all coefficients in an insignificant tree are known after coding it as a zerotree.

B. On Wavelet Coefficients

Compression works by finding how the input data are correlated, and then by removing this correlation so that the data can be represented in fewer bits. In order to explain why a particular wavelet image coder performs well, we need to understand some of the statistical properties of wavelet coefficients so that we can see how the image coder exploits them. These properties and models of how wavelet coefficients are related to each other led to the design of algorithms such as EZW and SPIHT. This section discusses some of the properties of wavelet coefficients and shows how recent image coders use the properties for efficient image compression.

One basic property of wavelet coefficients is that most are zero or close to zero in value, and few coefficients have a large magnitude. Another is Shapiro's *zerotree property*, which states that if a coefficient is found to be insignificant, then the entire tree rooted at that coefficient is also likely to be insignificant. These properties ensure that zerotrees are likely to be common, and thus motivated the design of zerotree coding.

Another property of wavelet coefficients is that the magnitude of a child coefficient is usually less than its parent. This implies that coefficients in a lower frequency subband are more likely to become significant before coefficients in a higher frequency subband. Shapiro's EZW algorithm utilizes this knowledge when it subdivides the significant trees that it finds: EZW always subdivides a significant tree into several parts, where the first part is former root coefficient of the significant tree. This root coefficient is always checked for significance before the other parts of the significant tree.

The key observation in SPIHT was that if a coefficient becomes significant in bit-plane k, then it is likely that its siblings (those other coefficients having the same parent) will become significant in bit-plane k or became significant in a previous bit-plane. By exploiting this fact in the way it subdivided its trees, SPIHT improves upon EZW in terms of rate distortion performance.

More recent work ([6], [3], [7], [8]) has shown that significant coefficients are likely to be found in clusters that are close together. This is similar to the property that SPIHT exploited, but more general because it implies an adjacent neighbor of a significant coefficient in any subband is likely to be significant, regardless of whether the two coefficients share the same parent. This can be exploited by altering the strategy of coding a coefficient based on the values of its neighboring coefficients.

In general, the more correlation between coefficients that an algorithm can exploit, the better its compression efficiency. We can see the basic trend in obtaining better compression is to find some property of wavelet coefficients that was not previously accounted for, and then to exploit it to obtain better compression.

III. Group Testing Background

A. Introduction

The concept of group testing was originally derived from the problem of identifying Army recruits that were infected with syphilis [5]. A laboratory blood test could detect the presence or absence of syphilitic antigen. Instead of testing the recruits individually, the blood samples of several men could be pooled together and tested. A lack of the antigen would imply none of these men have syphilis; its presence would imply at least one of these men has syphilis. If the percentage of infected recruits is small, then pooling the bloods samples can greatly reduce the required number of laboratory tests.

A simple version of the group testing problem can thus be defined as follows: Given n items, s of which are significant, what is the best way to identify the s significant items? It is assumed that items can be identified as significant or insignificant only through *group tests*. A group test is the process of picking a subset of the n items and determining whether there is a significant item in that set. There are two possible outcomes of a group test on set K: either K

is *insignificant* (meaning all items in K are insignificant), or K is *significant* (meaning there is at least one significant item in K).

Algorithms solving this and many other group testing problems can be found in Du and Hwang's book [5]. Most of this book focuses on combinatorial group testing, where the objective is to minimize the number of group tests that can occur in the worst case. In probabilistic group testing literature (see [9]), a probability distribution is put over the set of items and the objective is to minimize the expected number of group tests.

B. Applicability

Group testing has been found to be widely applicable to problems in many different areas. Besides screening for disease, applications have been found in areas as diverse as industry (for identifying defective Christmas lights), graph theory, and fault tolerant computing. In fact, group testing can also be used for data compression, if we view compression in the following manner:

Given a binary bit stream as input, think of each input bit as an item, where 1's represent significant items, and 0's represent insignificant items. Now let an encoder use a group testing algorithm to identify the significant items (the 1's in the bit stream). As output, it would send binary bits representing the outcomes of the group tests performed: 0 for an insignificant set, and 1 for a significant set. As long as the method of choosing the input bits for each group test is deterministic, a decoder could infer the input bit stream based solely on the output.

Given this encoding and decoding scheme, the obvious next step is to ask how well a group testing algorithm would perform on compressing the given data. Since group testing algorithms try to minimize the number of group tests, in our scenario, that is equivalent to minimizing the number of bits in the encoder's output. Since this is exactly what a good compression scheme would do, we expect good group testing algorithms to also be good at compressing data.

Group testing can also be used for wavelet-based image compression. Recall (see section II) that recent image compression techniques code the wavelet coefficients bit-plane by bit-plane, and code whether a coefficient is significant or insignificant. These bits can be directly coded with any group testing algorithm.

In fact, group testing can be thought of as a generalization of the zerotree coding technique. Recall that zerotree coding tests trees of coefficients for significance, where trees are significant only when some coefficient in that tree is significant. This is exactly the same as performing a group test on the coefficients in that tree. The significance pass (called the sorting pass in [2]) of the SPIHT algorithm can be thought of as a group tester that identifies the significant coefficients of a given bit-plane. Group testing is more general than zerotree coding because the groups that it tests together are not restricted to be trees, but can be arbitrary collections of coefficients.

C. Group Testing Algorithms

C.1 Hwang's Algorithm

One of the first group testing algorithms was proposed by Hwang in 1972 [10]. As input, the algorithm is given n items and the knowledge that s of these n items are significant. The algorithm identifies the s significant items as follows. Let P be the initial set of n items. Select a subset K of P of size k (the choice of k is critical and will be discussed later). Test whether K is significant. If K is insignificant then we repeat the process for the set $P - K$ with the knowledge that all the members of K are insignificant. If K is significant then the algorithm uses *binary splitting* (described below) to identify a single significant item x in the set K. Binary splitting will also identify a set I of insignificant items, where $I \subset K$. The algorithm then repeats the process, trying to identify $s-1$ significant items in the set $P - (I \cup \{x\})$. The algorithm terminates when s significant items have been identified. Any untested items are then determined to be insignificant. The process of selecting the set K for testing is called a *group iteration*.

The binary splitting algorithm works on an input set K of size k, with K assumed to be significant (so that at least one item in K is significant). If $k = 1$ we are done. Otherwise, partition K into two roughly equal sized parts, K_1 and K_2. Perform a group test on K_1. If K_1 is insignificant, then we know K_2 is significant, so recursively search K_2 for a significant item. Otherwise, K_1 is significant, and K_1 can be recursively searched for a significant item. Eventually, a significant item will be found, and it will take at most $\lceil \log_2 k \rceil$ tests to find a significant item in K. In the process, all the items in the sets that were tested to be insignificant are identified as individually insignificant.

The choice of the group iteration size k can drastically affect the effectiveness of the group tester. Choosing k to be large is helpful if the result is insignificant; in this case all k items will be identified as insignificant. However, choosing k to be small is helpful if the result is significant; in this case it will not take many tests to identify which item is significant. We want to choose k to strike a balance between these two cases. Intuitively, a good choice for k would be one where the probability that our set K is significant is about $1/2$, so that half the time k items are identified as insignificant, and the other half of the time we look for a significant item which does not take too many tests to find. Hwang chose $k = 2^{\lfloor \lg \frac{n-s+1}{s} \rfloor}$, where n is the number of items left, and s is the number of significant items left. Note that it is a power of 2 to simplify binary splitting.

Since we do not require powers of 2 for our work, our method of choosing k is different from Hwang's and based on entropy arguments found in section III-C.3. Our exact choice can be found in section IV-D.2.

C.2 Example

Consider a binary source where each bit has value 0 independently with probability $p = .92$, and apply Hwang's group testing algorithm. For this value of p, a good choice

for the group iteration size would be $k = 8$, because the probability that one of our sets is insignificant is $.92^8 \approx \frac{1}{2}$. For this example, we assume that when k items are chosen, they are the first k bits in the source that are not yet coded. We also assume that whenever a set K is partitioned into two sets, the first set tested (K_1) will contain the bits of the first half of K, in the order that the bits were in the input source.

We will first examine two cases, one where the first 8 input bits are `00000000` and the other where they are `00000110`. We will use `?` to denote bits whose value are not yet coded. Figure 2 shows the sequence of group tests performed, and the resulting output bits for the two example cases.

State of Input	Output Bit
Case I:	
[? ? ? ? ? ? ? ?]	0
0 0 0 0 0 0 0 0	
Case II:	
[? ? ? ? ? ? ? ?]	1
[? ? ? ?] ? ? ? ?	0
0 0 0 0 [? ?] ? ?	1
0 0 0 0 [?] ? ? ?	0
0 0 0 0 0 1 ? ?	

Fig. 2. Example showing group tests performed. Each box represents a group test of the items inside the box.

We can calculate the output produced by a group iteration of size 8 by examining all possible cases; the result is shown in table I. If the 8 input bits are all `0`, then one group test (and thus one output bit) is used to describe all 8 bits. In all other cases, the first output bit signifies that the set of input bits is significant, and the remaining 3 output bits indicate the position of the first significant input bit. The value of the input bits represented with a '`?`' will not affect the output bits sent. Note that the value of these bits will remain unknown after one group iteration; their value must be determined in a subsequent group iteration. In contrast, the input bits represented with '`0`' will be coded (known to be insignificant) after one group iteration.

Table I also shows that Hwang's algorithm essentially yields a variable-to-variable length code. If we ignore the `?`'s in the table, then the string of eight `0`'s codes to the single bit 0 while each of the other run-length strings ending in `1` codes to a 4 bit string. This is the elementary Golomb code of order 8 (see [6]) which is equivalent to a Golomb code [11] of order 8 applied to the sequence of zero-run lengths. Generalizing, we see that the code that a group iteration of size k produces is equivalent to an elementary Golomb code of order k. Interestingly, Ordentlich, *et al.* [6] and Malvar [12] both use elementary Golomb codes to effectively code wavelet coefficients. Because zerotree coding and Golomb coding are both equivalent to group testing we can state that a significant number of wavelet coding techniques essentially rely on group testing.

TABLE I
GROUP TESTING CODE WITH GROUP ITERATION SIZE 8

Input bits	*Output bits*
`00000000`	0
`00000001`	1000
`0000001?`	1001
`000001??`	1010
`00001???`	1011
`0001????`	1100
`001?????`	1101
`01??????`	1110
`1???????`	1111

C.3 Entropy

One goal of codeword design for lossless compression is to achieve an average code length that is as close to the entropy of the source as possible (that is, to have low redundancy). Although minimizing the number of group tests in our group testing framework also minimizes the length of the code generated by group testing, it does not tell us how well group testing performs relative to the entropy.

Gallager and Van Voorhis [13] studied the effectiveness of Golomb codes in terms of entropy, for memoryless sources. By directly applying their results, we know that the optimal group size k is the unique integer k satisfying the inequalities

$$p^k + p^{k+1} \leq 1 < p^k + p^{k-1}, \tag{1}$$

where p is the probability that a source bit has value `0`. It can be shown that for this choice of k, elementary Golomb codes, and therefore group testing using Hwang's algorithm, is always within 5% of entropy, and usually much lower. Figure 3 compares the bit-rate due to group testing against the entropy of a source, for various values of p.

Note that when k is not a power of 2, then it is not immediately obvious how binary splitting should partition the initial set of K of size k into two sets K_1 and K_2 in order to minimize the expected number of bits required to code the group iteration. It turns out that we should partition K into two sets of size i and $k-i$ where i is a power of two and $k-i$ is as close to i as possible. Furthermore, the set of smaller size should be tested first. Making this choice in the binary splitting is the only way to ensure that group testing with group iteration size k is the the same as an elementary Golomb code of order k for all values of k. The reason for this choice is explained below.

The example in section III-C.2 shows that group testing with group iteration size k is a variable-to-variable length code where the input sequence 0^k is coded with 0, and for $0 \leq j < k$, 0^j1 is coded with a codeword of that starts with a 1. Define $l = \lfloor \log_2 k \rfloor$. If k is a power of 2, then $l+1$ is the length of the codeword for the input sequence 0^j1. When k is not a power of 2, then of the k input sequences

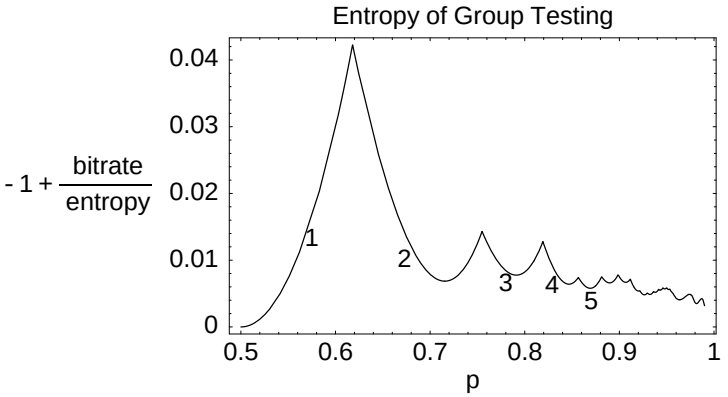

Fig. 3. Compares group testing bit-rate to entropy, on a memoryless source where p is the probability that a source bit is 0. Numbers on the graph indicate the optimal group iteration size for that section of the curve.

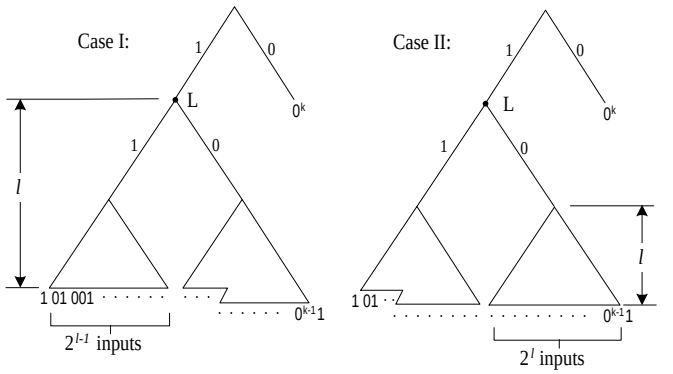

Fig. 4. Binary tree representing the codewords produced by Hwang's group testing algorithm. Left branches and right branches correspond to output bits of 1 and 0, respectively. Leaves represent the input sequence that was coded on that branch.

of the form 0^j1, $k' = 2^{l+1} - k$ of them will be coded with a codeword of length $l+1$, while $k - k'$ of them will be coded with a codeword of length $l+2$. Choosing the sizes of K_1 and K_2 corresponds to deciding which of the input sequences 0^j1 will have longer codewords than the others. Clearly, we want the less probable input sequences to have the longer codeword, and more probable input sequences to have shorter codewords. If p is the probability that a source bit is zero, then the probability of input sequence 0^j1 occurring is $p^j(1-p)$. This implies that if $j < k'$, we should code 0^j1 with a codeword of length $l+1$, and that the other input sequences should be coded with codewords of length $l+2$.

We can use a binary tree to represent the series of group tests that the binary splitting algorithm makes, as illustrated in figure 4. The root node represents the initial group test of size k. Every other node represents a group test of size equal to the number of leaves in the subtree rooted at that node. Each left branch signifies that the group tested was significant, and each right branch signifies that the group tested was insignificant. Let node L be the left child of the root node, and consider the subtree rooted at this node. In this subtree, the left-most leaf represents a series of group tests where the group tested was always significant. This corresponds to where the input sequence was 1. The right-most leaf of this subtree corresponds to a series of group tests where the each group tested was insignificant. This corresponds to the input sequence $0^{k-1}1$. In fact, the leaves at the bottom of the tree represent the input sequence $1, 01, 001, ..., 0^{k-1}1$ when read from left to right. Furthermore, the depth of a leaf represents the length of the associated output codeword. Thus, we would like to construct the tree where the first k' leaves are at depth $l+1$, and the rest of the leaves are at depth $l+2$

Recall that in the binary splitting algorithm, we split set K of size k into two sets K_1 and K_2, respectively of size k_1 and k_2, and test set K_1 first. If K_1 is significant, then we will have to identify a significant item from a set of k_1 elements; if K_1 is insignificant, we will have k_2 elements from which to identify a significant item. This implies that the left child of node L has k_1 leaves, and that the right child has k_2 leaves. To optimize the depths of the leaves for coding performance, either the left child of node L should be a complete binary tree (case I), or the right child of node L should be a complete binary tree (case II). This implies one of K_1 or K_2 must have a power of 2 number of elements. Furthermore, we also need to test the smaller set first so that number of elements on the left branch of node L will be less than the number of elements on the right branch. This is the reason behind our choice of partitioning during binary splitting.

C.4 Other Group Testing Algorithms

Hwang's group testing algorithm assumes the number of significant items in the initial set is already known. This number is used to select the group iteration size k. In the context of data compression, the encoder can send this number as side information to the decoder at the beginning of the bit stream. However, it may be preferable to not send this information at all, but instead estimate k adaptively as bits are encoded. Strategies for choosing k can be found from both competitive group testing literature [14] and adaptive Golomb coding literature [15].

C.4.a Competitive Group Testing. In competitive group testing, the problem of group testing is solved when s, the number of significant items, is unknown. The goal is to minimize the ratio of how an algorithm performs in the worst case compared to how well the best algorithm that knows s would do. One example is the doubling strategy of Bar-Noy *et al.* [14], where k, the size for each group iteration, starts at 1, and doubles every time. It continues doubling as long as no significant items are found, and resets to 1 once a significant item is found. Note that the goal of competitive group testing does not directly minimize the number of group tests, and thus the algorithms in this area may not be as directly applicable to data compression.

C.4.b Adaptive Golomb Coding. Adaptive Golomb coding could also be used to pick k. The algorithm of Langdon [15] starts out with $k = 1$. Then k adapts according to the result of the previous group iteration: if no significant item was found, then k doubles, otherwise k halves in value (although k never drops below 1). This simple adaptive strategy was meant to adapt very quickly to the input source, as well as to be very inexpensive computationally.

IV. The GTW Algorithm

A. Overall Algorithm Design

The design goal behind GTW is to find the most effective method of using group testing for image compression. Recall that the significance pass in SPIHT is really one method of implementing group testing. We thus propose a new image coder that replaces SPIHT's significance pass with a different method of group testing; this new method will be based on Hwang's group testing algorithm. Other aspects of our image coder, such as the wavelet transform and the refinement passes will be the same as that of SPIHT.

An obvious way of using group testing in the significance pass is to simply take all the coefficients in order, from lowest frequency subband to highest frequency subband, and code them with a group testing coder. Unfortunately, this will not work well because of the assumptions made in the group testing problem. In particular, the group testing problem assumes that nothing is known about the initial items. This is similar to assuming that all coefficients are equally likely to be significant, and that all coefficients are independent of each other. As explained in section II-B, neither of these facts is strictly true in our application for wavelet coefficients.

One way to surmount this difficulty is to make the characteristics of our input to group testing match with what the group testing algorithm expects. This means that when we perform a group iteration on a set of items, we want each item in the set to be independent of the others, and we want each item in the set to be equally likely to be significant. To do this, we will organize all the coefficients into *classes* that tend to have our desired properties. We will try to make the coefficients in each class as independent as possible of each other, as well as approximately equally likely to be significant. Each class could then be coded with a different adaptive group tester; this is necessary because we expect different classes to have different probabilities of being significant.

Putting coefficients into classes is similar to choosing different contexts for the coefficients, and coding each context with a different entropy coder. Thus, we must design our classes with the context dilution problem in mind. That is, having too many classes will result in many classes with only a few coefficients. Group testing may never effectively adapt to these classes, resulting in poor coding performance. On the other hand, having many classes may be beneficial because the coefficients in a class will have similar characteristics. This will lead to better adaptation and coding performance. The number of classes must be chosen to balance the effectiveness of many classes with the ineffectiveness of very few coefficients in each class.

B. GTW Classes

We define the *GTW classes* based on the properties of typical wavelet transform coefficients of image data; these properties were derived from previous work such as SPIHT. The characteristics that distinguish between GTW classes are the *subband level*, the *significant neighbor metric*, and the *pattern type*. We now proceed to describe these characteristics.

B.1 Subband Level

In a bit-plane of a wavelet-transformed image the coefficients in lower frequency subbands are more likely to be significant. The lowest frequency subband counts as a subband level. There is one additional subband level for each level of the wavelet transform. Figure 5 shows the 4 subband levels when 3 levels of the wavelet transform are performed.

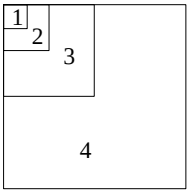

Fig. 5. Illustration of subband levels of a wavelet transformed image.

B.2 Significant Neighbor Metric

A coefficient in a bit-plane is likely to be significant if more of its *neighbors* are significant. A neighbor is defined to be the following: one of up to 8 spatially adjacent coefficients in the same subband, one of the 2 spatially identical coefficients in another subband at the same level, the parent coefficient in the next lower subband, or one of the 4 child coefficients in the next higher subband. The parent/child relations are exactly as defined in SPIHT. Figure 6 shows the neighbors of a coefficient. There are 4 values in the significant neighbor metric, `0`, `1`, `2`, and `3+`, corresponding to whether 0, 1, 2, or more than 2 neighbors are significant, respectively. In counting the significant neighbors we count one for each spatially adjacent, spatially identically, and parent coefficients, but just one for all 4 children coefficients. That is, if one or more children are significant then we add one to the count of significant neighbors. Thus, the maximum neighbor count is 12 even though there are 15 neighbors. In determining the significant neighbor metric for a coefficient in the current bit-plane, a neighbor can be known to be significant from the coding of a previous bit-plane or from the coding of the current bit-plane.

We derived our significant neighbor metric from several considerations. We chose to limit the 4 children to count as at most one because the 4 children, taken together, represent one spatially equivalent spot in the next higher subband level. Also, knowing that 4 child coefficients are sig-

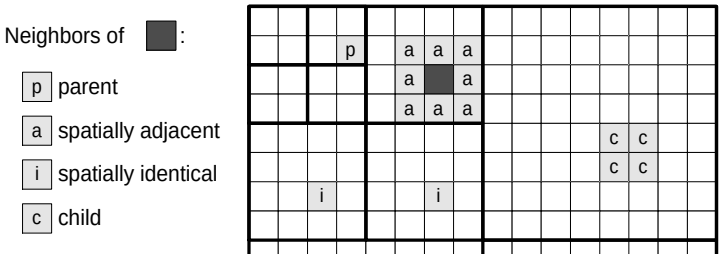

Fig. 6. Illustration of neighbors.

nificant does not seem to bear 4 times as much information as knowing that one child coefficient is significant.

We reduce the resulting count of significant neighbors to four values for several reasons. One is too avoid having too many classes, and suffering from a context dilution problem. Another is the observation that once the metric for a coefficient reaches about 4, then the probability of significance for that coefficient is high enough that a group iteration size of one is sufficient. The inequalities (1) show that $k = 1$ when $p \leq (\sqrt{5}-1)/2 \approx .62$, implying that when the probability of being significant is greater than about .38 then a group size of 1 is optimal.

B.3 Pattern Type

Coefficients adjacent to each other are assigned different pattern types. The pattern type is based solely on position in a subband. In GTW, each coefficient belongs to one of 4 distinct pattern types as shown in figure 7.

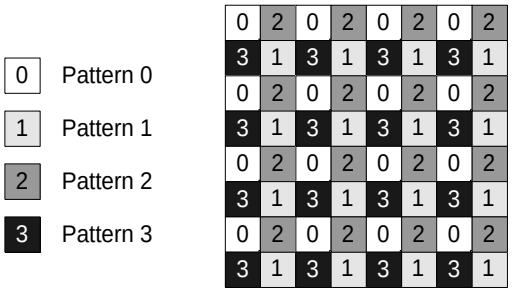

Fig. 7. Illustration of pattern type.

One idea behind using the pattern type is to make coefficients in any class less likely to be correlated. This will make the coefficients more likely to be independent of each other, which is what Hwang's group testing algorithm expects.

We can also view the pattern type as an attempt to control the order in which the information known about neighboring coefficients propagates. Let C_i represent the set of coefficients with pattern type i. If we assume all coefficients in C_i are coded before those in C_{i+1}, then figure 8 shows that a coefficient in C_{i+1} knows more information about its neighbors than in C_i. The greater amount of available information should make it easier to code C_{i+1} than C_i. We call this the *bootstrapping* effect. Controlling the coding order of the pattern type will change how accurately the coefficients in C_i are classified, and how well they are coded. This may help improve the net coding efficiency.

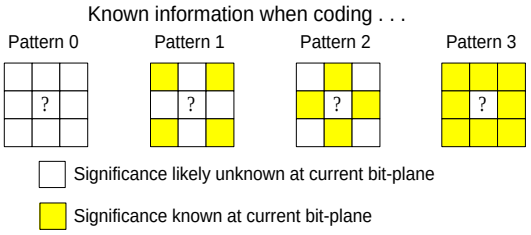

Fig. 8. Illustration of bootstrapping.

C. Class Ordering

A GTW class is defined by these three characteristics: subband level, significant neighbor metric, and pattern type. For our study we used 7 subband levels, 4 significant neighbor metric types, and 4 pattern types for a total of 112 classes. With these specified classes, what is the best order in which to code these classes? From the perspective of generating an embedded bit stream, we should code the coefficients that contain the most information first. Since finding significant coefficients increases the fidelity of the image and finding insignificant coefficients has no effect on the fidelity, those coefficients more likely to be significant should be coded first.

Based on this heuristic, we should code the low subband level classes before the high subband level classes. We should also code the classes with more significant neighbors before those with fewer or no significant neighbors. After some testing (see section VI-C), we determined that the following ordering resulted in good rate-distortion performance:

GTW Ordering: Always code the classes with the greatest significant neighbor metric first. If there are several classes with the same significant neighbor metric, then code the classes with the lowest pattern type first. If there are classes with the same significant neighbor metric and pattern type, then code the class with the lowest subband level first.

One way to verify that this ordering works well is through figure 9. It is a scatter plot of the log of the group iteration size k for each class. Every group iteration performed is placed in a rectangular bin. Darker bins correspond to bins containing more elements. Classes are ordered according to the GTW ordering; thus, the classes on the left of the plot are always coded before the classes on the right. This graph shows that the classes with the lower group iteration sizes tend to be coded first. The classes with low group iteration size also tend to contain those coefficients most likely to be significant.

Note that the ordering depends mostly on the significant neighbor metric because it happens to be the best predictor for when a coefficient is significant.

In addition to improving the image quality, finding significant coefficients quickly will contribute to better classification of their neighbors. Once a significant coefficient is

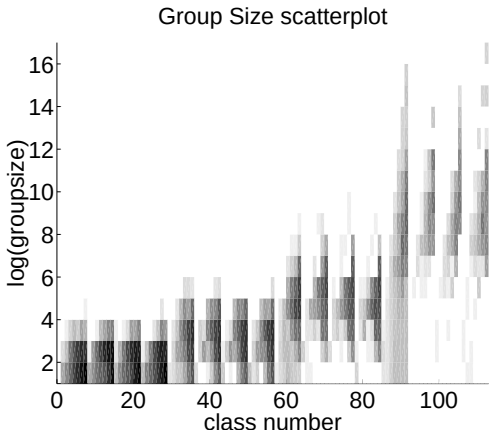

Fig. 9. Group iteration size scatter plot.

found, its neighbors could change classes because they now have one more neighboring coefficient that is significant. The new classes will be coded earlier than the old classes that the coefficients were formerly contained in. This has an effect of searching for clusters of significant coefficients because once a significant coefficient is found, then neighboring coefficients are more likely to be coded next. If any of these neighboring coefficients were tested as significant, then the coefficients around them will also be more likely be coded earlier.

D. GTW Details

In our study we perform 6 levels of wavelet transform using the Daubechies 9/7-tap filters [16] on 512×512 gray-scale images. In this case, there are 7 subband levels leading to an 8×8 lowest frequency subband. The average of the coefficients in the lowest frequency subband is subtracted from the lowest frequency subband before the significance and refinement passes start. This average is coded and sent in the first bits of GTW. The refinement pass works exactly the same as the one in Said and Pearlman's SPIHT. The significance pass and the adaptive group testing strategy are described below.

D.1 Significance Pass

The GTW algorithm to code the significant coefficients in a bit-plane is fairly simple. The basic idea is to repeatedly code coefficients in the GTW classes until all have been coded. Each class has its own group size statistic. These statistics are used to calculate the size of the next group iteration for that class, according to our adaptive group tester defined in section IV-D.2.

We try to code the coefficients in a class only when the GTW class is *adequate*. A class C is adequate if there are at least k_C coefficients in that class, where k_C is the current group iteration size for class C. Coding only adequate classes ensures that group testing might find k_C insignificant coefficients with one test; this is necessary for good coding performance.

Our Algorithm works as follows:
Repeatedly:

1. Perform a group iteration on the group within the first adequate GTW class, according to the GTW ordering of the classes. Output bits corresponding to the group iteration.
2. If a significant coefficient is found, output its sign and update the neighbors of that coefficient (coefficients could change classes at this point).
3. Update the group size statistics for the class tested.

At the end of this loop, there will be a few coefficients left in inadequate classes. At this point, we start coding the coefficients in the inadequate classes. We combine the coefficients from different classes together and code these combined groups. In this context, a group of coefficients is *end-adequate* when it contains k coefficients, where k is the miniumum k_C such that a coefficient of the group is a member of class C.

We can view the remaining coefficients as being on a list ordered by the GTW ordering. Our method chooses the first group of coefficients in this list that is end-adequate, and performs a group iteration on that group. If there are not enough remaining coefficients to form an end-adequate group, then all these coefficients are combined into one group and tested together. This process is then repeated until all coefficients are coded.

D.2 Adaptive Group Testing Strategy

The only difference between our group testing strategy and Hwang's algorithm is in our adaptive method of choosing the group iteration size k. Since our k does not have to be a power of 2, our implementation of binary splitting is slightly more general than Hwang's. As described in section III-C, binary splitting partitions a significant set K into two sets K_1 and K_2 of roughly equal size. Our implementation of binary splitting chooses the cardinality of the two sets to be i and $k-i$ where i is a power of two and $k-i$ is as close to i as possible. The size of K_1 is chosen to be $\min(i, k-i)$. For example, a set of size 11 is partitioned into sets of size 4 and 7 with the set of size 4 tested first, and a set of size 13 is partitioned into sets of size 5 and 8 with the set of size 5 tested first.

Our adaptive strategy for choosing the group iteration size k_C for class C works as follows: Continually keep track of s_C, the number of significant coefficients identified so far in class C, and n_C, the total number of coefficients identified so far in C. Initially, when $n_C = 0$, start with group iteration size $k_C = 1$. While no significant coefficient is found in the previous group iterations for C, double k_C and perform another group iteration. Once the first significant coefficient is found, use $p = (n_C - s_C)/n_C$ as the probability estimate of insignificance, and choose k_C satisfying the inequalities (1). Note that the initial phase of our strategy is the same as both the doubling strategy taken from Bar-Noy, *et al.* [14] and Langdon's [15] adaptive Golomb coding method. This scheme quickly adapts to the characteristics of the source.

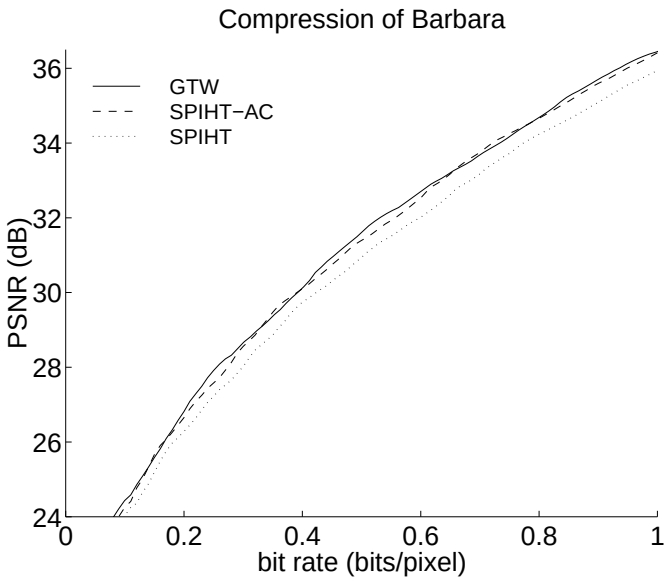

Fig. 10. Comparison of SPIHT and GTW.

TABLE II
COMPARISON OF GTW AND SPIHT-AC. Δ SPIHT-AC = SPIHT-AC − GTW.

Image	*Algorithm*	*Rate* (Bits/pixel)			
		0.1	0.25	0.5	1.0
rough wall	GTW	24.37	27.22	29.51	32.51
	Δ SPIHT-AC	+.31	+.10	0.00	−.07
couple	GTW	26.12	29.13	32.41	36.45
	Δ SPIHT-AC	+.05	+.08	+.04	+.13
man	GTW	27.80	31.31	34.13	37.42
	Δ SPIHT-AC	+.32	+.05	+.12	−.15
boat	GTW	27.31	30.93	34.27	39.01
	Δ SPIHT-AC	+.05	+.04	+.18	+.11
tank	GTW	27.50	29.35	31.17	33.86
	Δ SPIHT-AC	−.05	+.01	+.01	−.08
Goldhill	GTW	27.86	30.49	33.09	36.42
	Δ SPIHT-AC	+.08	+.07	+.04	+.13
Lena	GTW	30.17	34.17	37.28	40.45
	Δ SPIHT-AC	+.05	−.06	−.07	−.04
Barbara	GTW	24.41	27.86	31.49	36.42
	Δ SPIHT-AC	−.15	−.28	−.09	−.01

V. Results

We present our results on the 512 × 512 Barbara image and compare our PSNR curve with the ones for SPIHT with and without arithmetic coding. Figure 10 shows that GTW performs about 0.7 dB better than SPIHT and about the same as SPIHT-AC. We chose to examine Barbara because it is harder to compress than an image like Lena. The qualitative results found for Barbara are also present in other images. However, quantitatively, the results are more pronounced for Barbara; this makes it easier to see the difference between two algorithms.

Table II compares GTW against SPIHT with arithmetic coding for a wide variety of natural images. All images are 512 × 512 in size, except for the man image which is 1024 × 1024. All are 8-bit gray-scale images. The standard test images Goldhill, Lena, and Barbara can be obtained from UCLA's web site [17]. The other images are part of the USC image database [18]. All results are measured in terms of PSNR, in dB. The Δ SPIHT-AC rows represent the difference in PSNR between SPIHT-AC and GTW. Positive numbers indicate the amount of PSNR improvement of SPIHT-AC over GTW. We can see that the difference between SPIHT-AC and GTW is very small.

VI. Alternative Approaches

A. Alternative Significant Neighbor Metric

It is clear that our significant neighbor metric is *ad hoc* in nature, since all neighbors of a coefficient c (except its children) have an equal say towards whether c is likely to be significant. By performing statistical analysis across an image, it should be possible to calculate a set of weights $w_1, w_2, ..., w_{15}$ for the 15 neighbors of c that represent how much each neighbor contributes to c's significance status. By taking the statistics into account, we are able to better predict whether c will become significant, resulting in better compression.

As a more "principled" approach, we redefined the significant neighbor metric to be $\sum_{i=1}^{15} w_i b_i$, where w_i is the weight given to the ith neighbor of a coefficient, and b_i is the bit value of the ith neighbor (1 for significant, and 0 for insignificant or unknown). Although the numeric weights could be calculated by many different methods, we used the logit [19] approach, which is a standard method in statistics for assigning weights when the training data are discrete. To compress an image, we calculated the weights using that image and then sent the weights in the initial part of the compressed bit-stream.

Figure 11 is a graph that shows the percentage of significant coefficients found in the set of coefficients that had a given value as its significant neighbor metric. Comparing with figure 12, it appears that the logit metric and the GTW significant neighbor count give about the same information. The logit metric is slightly better because at low values of the metric, the percentage of significant coefficients is higher. However, we feel the added complexity of training to obtain the weights was not worth the slight benefits. Overall, this approach performed slightly better than our significant neighbor metric. The improvements over the *ad hoc* method ranged from 0.0 dB to 0.15 dB at various bit rates on the Barbara image.

B. Alternative Pattern Types

We observed that the coefficients that just became significant in the current bit-plane had a substantial effect on the value of the significant neighbor metric for its neighboring coefficients. Thus, it would be nice to maximize the amount of information known about neighboring coefficients. Unfortunately, this may not be possible in an overall sense: We must pick a specific ordering in which to code the coefficients, and on average, when coding a specific coefficient, we expect roughly half of its currently insignificant neighbors to not yet be coded at the current bit-plane.

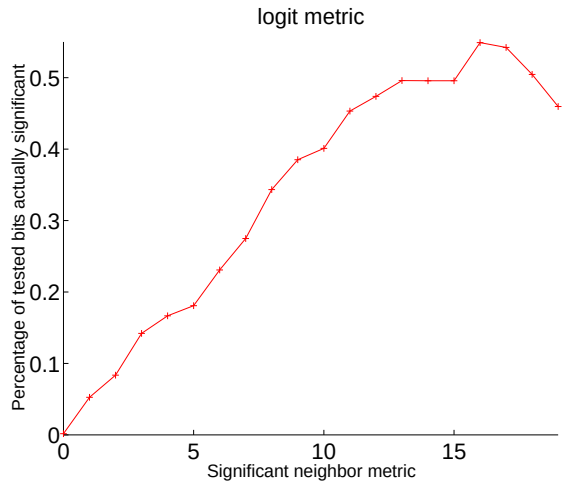

Fig. 11. Logit metric value uniformly quantized into 20 intervals, from training on Barbara.

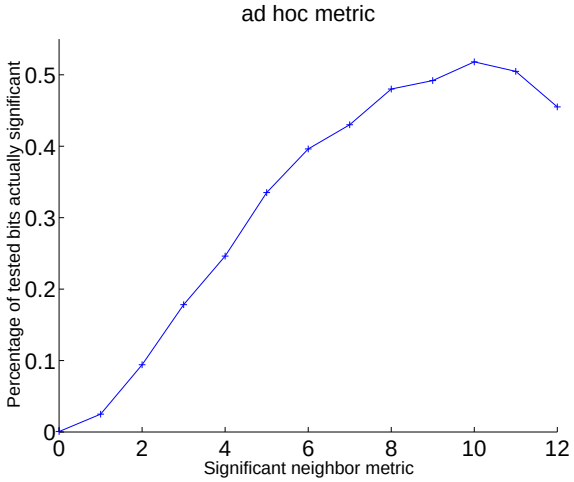

Fig. 12. Adhoc significant neighbor count in GTW, from Barbara.

Although maximizing the information known might not be possible, we can change the rate at which the information about neighboring coefficients increases. Recall that the bootstrapping effect describes how information increases as more coefficients are coded. This effect is different for different patterns, as illustrated in figure 13. We tried the two alternative patterns in figure 13 and found no difference in coding efficiency between these different patterns and the original pattern. The rate-distortion curves for the Barbara image using different pattern types looked almost exactly the same as that of the original pattern.

We also tried eliminating the pattern type completely, so that there were only 28 classes. This change performed slightly worse than GTW on the Barbara image. In fact, the PSNR curve for this method looked exactly the same as the method in section VI-C when the class ordering was significant neighbor metric first, subband level next, and the pattern type characteristic last.

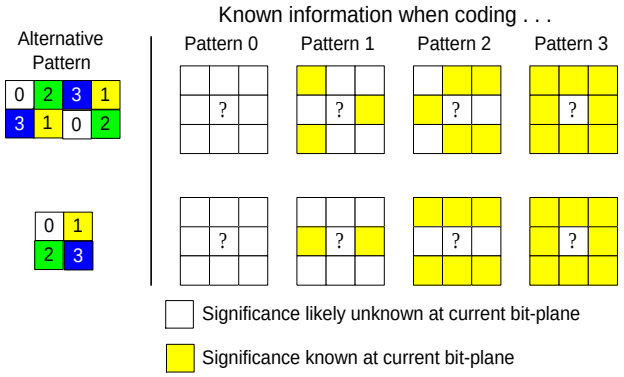

Fig. 13. How information propagates in different patterns

C. Varying Ordering of Testing Groups

C.1 Varying GTW Class Ordering

The class ordering in GTW, described in section IV-C, considers the 3 class characteristics in the following order: significant neighbor metric first, then pattern type, then subband level. We tried our algorithm with each of the 6 possible total orderings of the 3 different characteristics. From experimentation we found that these alternative orderings did not perform as well; figure 14 shows the performance of these orderings on the Barbara image.. Intuitively, it would appear that considering the subband level first would give good results because lower frequency subbands tend to have more significant coefficients in the early bit planes. However, ordering with subband level first produced concave dips in the rate distortion curve. This could mean that it pays off to identify significant coefficients in higher frequency subbands early. Furthermore, the orderings with pattern type first were significantly worse, because the pattern type is not useful for predicting significance of coefficients.

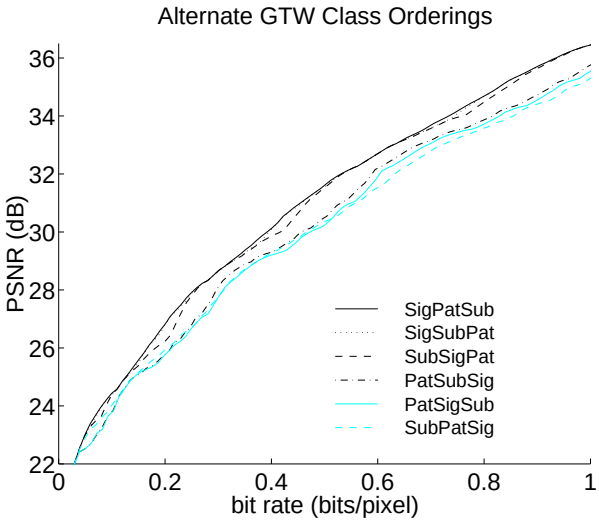

Fig. 14. Different class orderings on the Barbara image. Sig, Sub, and Pat stand for Significant Neighbor Metric, Subband Level, and Pattern Type, respectively. Each line is labelled by the order of importance of the characteristics, so that SigSubPat represents the normal GTW ordering.

C.2 Group Iteration Size Ordering

One of the reasons why ordering by significant neighbor metric first is better than ordering by subband level first is that the significant neighbor metric is better at predicting significance than subband level. In terms of rate-distortion theory, we would like to find the significant coefficients as soon as possible to reduce the distortion. Thus, it is better to test all the coefficients in order of most likely to be significant, to least likely to significant.

One way to apply this principle is to use the adaptive group testing statistics to set the ordering of which classes should be tested first. Classes with a smaller group size are more likely to have significant coefficients and thus should be tested first. We tried this ordering where classes were tested in the order of smallest group iteration size first. Ties were broken by the regular GTW ordering. On the Barbara image, the rate-distortion performance did not change with this technique.

C.3 Generalized Coefficient Ordering

We could also try different ways of ordering the coefficients within a particular class. Instead of arbitrarily group testing the first k coefficients that are in a GTW class, we could instead look for k most important coefficients in that class, and code them first. The criteria of importance could be based on the likelihood that a coefficient is significant, or on the amount of information a coefficient could give to its neighbors. This is the idea behind the two new orderings described in this section: the *parent-first ordering* and the *neighbors-first ordering*. Figure 15 shows how these orderings compare to the original GTW ordering on Barbara.

In the parent-first ordering, only coefficients whose parents are already coded are allowed to be tested in a group iteration. In this case, a class is considered inadequate if it does not have enough coefficients with coded parents to fulfill the group iteration size quota. The normal GTW ordering was used to decide which classes to check first. Note that with this ordering, it is still possible for coefficients in high frequency subbands to be coded before coefficients in low frequency subbands. This can happen when the coefficients in the high frequency subband are not descendants of the coefficients in the low frequency subband. This ordering performed slightly worse than the normal GTW ordering.

In the neighbor-first ordering, we allowed a coefficient of pattern type i to be coded only when its neighboring coefficients that have pattern type $< i$ are already coded. In this case, a class is considered inadequate if it does not have enough coefficients with an appropriate number of coded neighbors to fulfill the group iteration size quota. We can view this an attempt to maximize information flow, based on the following scenario: Suppose that c_a and c_b are both uncoded coefficients in the current bit-plane, and that all of c_a's neighbors are coded while none of c_b's neighbors are coded yet. Then it may be more beneficial to code c_b before c_a because c_b will provide information about itself to its neighbors, where as c_a will not. The neighbor first method performed significantly worse than the normal GTW ordering.

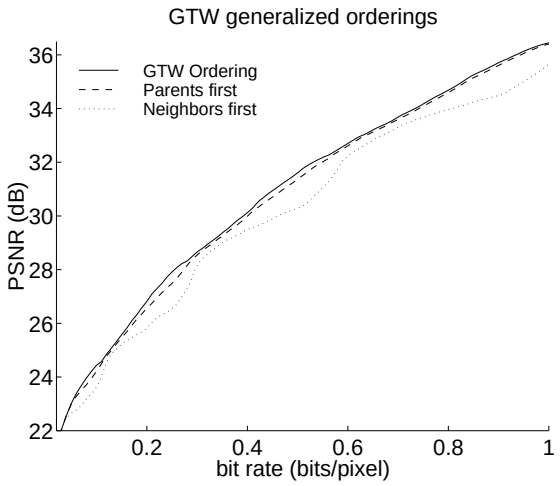

Fig. 15. Generalized coefficient orderings on the Barbara Image.

The results of these two orderings imply that trying to maximize the information flow between coefficients does not improve the net performance. Although better prediction can be obtained for coefficients when we maximize the information available, it appears that the slightly better prediction is not enough to offset the loss incurred from not coding the most likely to be significant coefficients first.

D. GTW with Arithmetic Coding

There is a significant gain in going from SPIHT to SPIHT-AC for Barbara. Such a gain does not seem possible in adding arithmetic coding to GTW. The SPIHT zerotrees take into account the correlation between a coefficient and its parent and children, but not its spatially adjacent coefficients in the same subband. Arithmetic coding in SPIHT-AC helps account for correlation between spatially adjacent coefficients by using different contexts for adjacent neighbors of different values. In GTW, spatially adjacent neighbor information is used as part of the context for group testing. Out of curiosity we added an adaptive first-order arithmetic coder on a binary symbol set to GTW. There was no appreciable difference in rate-distortion performance.

E. Varying group testing strategies

Another way to optimize GTW is to change the exact group testing strategy. Instead of using adaptive group testing, it is possible to initially send statistics saying what the group iteration size should be for each class. This method performed worse than our adaptive strategy.

VII. Conclusion

We have shown that the proposed method of coding GTW classes performs better than SPIHT's method of coding zerotrees, in the rate-distortion sense. This should not be surprising since GTW is a generalization of SPIHT. We

also have shown that it is possible to get performance comparable to SPIHT-AC without using arithmetic coding.

We conclude by highlighting the differences between GTW and other recent work in image compression. Ordentlich, Weinberger, and Seroussi's algorithm (OWS) [6] and Malvar's algorithm (PWC) [12] are similar to GTW in that they both use Golomb codes. The main differences are in what contexts are used and in what order the bits are coded. In OWS the coefficients are coded in a zig-zag order, where as in PWC the coefficients are reordered in a complicated but fixed way. GTW orders its coefficients dynamically, optimizing for information flow and for likelihood of being significant.

Wu's ECECOW algorithm [3] does not use group testing, but strict arithmetic encoding with context. Training is used to calculate the contexts that will be used in the coding. ECECOW has impressive rate-distortion performance, certainly better than SPIHT-AC and GTW, but neither SPIHT-AC nor GTW rely on training to achieve their results.

Chai *et al*'s SLCCA [8] coder identifies significant coefficients in clusters. It initially codes the starting positions of a cluster of significant coefficients, and expands a cluster by coding the coefficients around the starting position. When a significant coefficient's children is the start of a new cluster, then a significance link symbol is sent, so that the position of the new cluster need not be sent. This approach of looking for clusters of significant coefficients is similar to how GTW updates the neighbors of a significant coefficient, and codes the coefficients with the highest significant neighbor metric. Including the parent in the significant neighbor metric helps "link" parent clusters to their children clusters.

Taubman's EBCOT coder introduced the idea of having independent, finely embedded bit-streams that could be reordered in a user-defined fashion. The bit-plane coding technique uses a combination of zerotree coding, run-length coding, sign coding, and magnitude refinement. Each of these techniques is followed by arithmetic coding with carefully chosen contexts. They achieve better performance than GTW mainly due to the sign coding, and through keeping track of how the orientation of a subband affects the correlation between neighbors. Furthermore, they use the idea of "fractional bit-planes" to determine the order in which coefficients are coded. This is where they code the coefficients in a bit-plane in several passes. The coefficients coded in the earlier passes are those that are more likely to be significant. This method is similar to how GTW chooses the ordering of its GTW classes to find significant coefficients quickly. Since GTW propagates the information about significant neighbors as soon as they are discovered, we can view GTW as making many fractional bit-plane passes. This is in contrast to EBCOT which has only 4 passes in its fractional bit-plane method.

We believe that group testing is useful for its generality and applicability. This method unifies into a single framework many previously proposed wavelet coding algorithms, including zerotree algorithms and Golomb code algorithms. Furthermore, we believe this method is very flexible because it can be easily tweaked to efficiently code many different data sources. More sophisticated group testing approaches may lead to image compression algorithms with even better rate-distortion performance.

References

[1] J.M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, December 1993.

[2] A. Said and W. Pearlman, "A new fast and efficient image codec based on set partitioning in heirarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, June 1996.

[3] X. Wu, "High-order context modeling and embedded conditional entropy coding of wavelet coefficients for image compression," in *Thirty-First Asilomar Conference on Signals, Systems and Computers*, 1998, vol. 23, pp. 1378–82.

[4] R. Dorfman, "The detection of defective members of large populations," *Annals of Mathematical Statistics*, vol. 14, pp. 436–440, 1943.

[5] D. Du and F. Hwang, *Combinatorial Group Testing*, World Scientific, 1993.

[6] E. Ordentlich, M. Weinberger, and G. Seroussi, "A low-complexity modeling approach for embedded coding of wavelet coefficients," in *Proceedings Data Compression Conference*, March 1998, pp. 408–417.

[7] D. Taubman, "High performance scalable image compression with ebcot," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[8] B. Chai, J. Vass, and X. Zhuang, "Significance-linked connected component analysis for wavelet image coding," *IEEE Transactions on Image Processing*, vol. 8, no. 6, pp. 774–784, June 1999.

[9] M. Sobel and P. A. Groll, "Group testing to eliminate efficiently all defectives in a binomial sample," *Bell System Technical Journal*, vol. 38, pp. 1179–1252, 1959.

[10] F. Hwang, "A method for detecting all defective members in a population by group testing," *Journal of the American Statistical Association*, vol. 67, pp. 605–608, 1972.

[11] S. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, pp. 399–401, July 1966.

[12] H. Malvar, "Fast progressive wavelet coding," in *Proceedings Data Compression Conference*, March 1999, pp. 336–343.

[13] R. Gallager and D. Van Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Transactions on Information Theory*, vol. 21, pp. 228–230, March 1975.

[14] A. Bar-Noy, F. Hwang, H. Kessler, and S. Kutten, "A new competitive algorithm for group testing," *Discrete Applied Mathematics*, vol. 52, pp. 29–38, July 1994.

[15] G. G. Langdon, Jr., "An adaptive run-length coding algorithm," *IBM Technical Disclosure Bulletin*, vol. 26, no. 7B, pp. 3783–3785, December 1983.

[16] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Transactions on Image Processing*, vol. 1, pp. 205–220, April 1992.

[17] UCLA Image Communications Lab, "images," http://www.icsl.ucla.edu/ ipl/psnr_images.html.

[18] USC Signal and Image Processing Institute, "image database," http://sipi.usc.edu/services/database/Database.html.

[19] G. S. Maddala, *Introduction to Econometrics*, Macmillan Publishing Company, second edition, 1992.